PMX Device Driver Services
API Specifications

Document ID : HILK029

Issue Number : Draft P

Issue Date : 23 July 1998

Author : Marc A Stevens

DOCUMENT HISTORY

Issue	Date	Changes / Comments
DraftA -001	05/MAR/97	First Issue
DraftA -002	06/MAR/97	GetDataAddresses now returns pointer to Status Reg pointer - Status reg now held in host memory not Frame buffer. Added highlander memory allocate/free functions.
DraftA -003	08/MAR/97	16 and 32 bit DLLs no longer have single entry point due to thunking deficiencies. Concatenated the get/set functionality of all timing registers into one API. Added DDSFreeDisplay API.
DraftB	27 March 1997	Update for release to NEC
DraftC	2 May 1997	Added GetParamsAddress API and modified the data addresses returned by GetDataAddresses
DraftD	11 July 1997	16bit API reworked to cater for multiple devices, additional APIs and specified in terms of register interface for access to VxD directly not via 16bit DLL. 32Bit API spec written.
DraftE	16 July 1997	DeviceIOControl Interface defined
DraftF	24 Sep 1997	Added (16bit APIs) Alloc/FreeHostMemory and GetPageFrameTable, drastically extended the display list management routines and added I2C read, write and reset functions (32bit APIs.)
DraftG	2 Oct 1997	Added DDSState() 32bit (service table only) API and modified PCICONFIG struct (affects DDSGetPCIInfo() API).
DraftH	23 Oct 1997	Added DDSGetCMInfo(), modified parameters and text of DDSEnumerateDevices() and expanded on DDSFlip().
DraftI	24 Oct 1997	Expanded DDSTimingRegs() to include set and get of current display mode default timing registers.
DraftJ	4 Nov 1997	Added DDSMapDevnodeToDevice and DDSGetDisplayListID APIs. All other APIs, previously taking a devnode as input parameter, modified to take a pointer to DUC. Modified DDSSetCursorPos API and expunged from

		existence a couple of APIs that have moved to the display driver miniVDD.
DraftK	6 Nov 1997	Changes to ATTRWIN and CREATEWIN
DraftL	11 Nov 1997	Reworked Display List Management APIs to make more intuitive for DirectX HAL writers.
Draft M	2 July 1998	Removed 16-bit API. Reworked SetMode API
Draft N	7 July 1998	Changed DUC structure
Draft O	23 July 1998	Heavily reworked, updated and revised!
Draft P	14 Apr. 99	Added Primary Core interface, new state change def and DLMANEnableDevice Api.

TABLE OF CONTENTS

1. PURPOSE OF DOCUMENT

This document details the 32 bit API provided by the PMX Display Manager API module. PMX Display Manager (formerly, PMX Display List Manager) supports the Windows 9x, NT4 and NT5 operating systems via three different code-build options resulting in three OS-specific modules.

2. OVERVIEW OF THE PMX SERVICES MODULE

Implemented as a VxD or NT kernel mode driver, with a 32 bit API provided through a service table interface, the PMX Display Manager (DLMAN) module provides functionality for the creation and manipulation of display modes for the display devices attached to a PMX adapter, as well as interrogation of the capabilities of such devices. This includes functionality for the creation, moving, resizing and general management of video windows.

The main clients for DLMAN are the display driver and DirectDraw although Smart Tools also uses DLMAN services albeit via the 2D Display Driver's escape code mechanism.

2.1. Terminology

A **Device** is an adapter card.

A **Unit** is a display device on an adapter card e.g. CRT, TV or LCD.

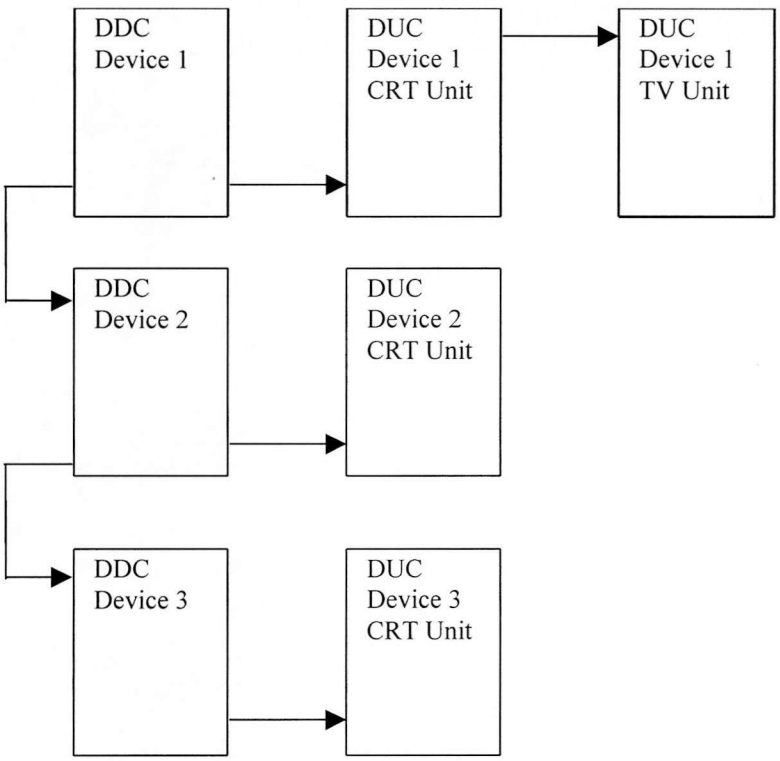
A **primary DEVNODE** is a DEVNODE created by Windows at device enumeration time which relates to a specific device.

A **subsidiary DEVNODE** is a DEVNODE created by Windows which is based on a primary DEVNODE for a device which supports more than one unit. A subsidiary DEVNODE is not available at initialisation time.

NOTE. This API is applicable for PMX1C and PMX1-LC.

3. API SPECIFICATION

DLMAN manages devices and units through DMDEVCTL (DDC) and DMUNITCTL (DUC) structures which are managed in a linked list. Each device has a single DDC and a DUC for each unit on that device. For example, in a system with three devices, the first with CRT and TV and the other two with CRT only, the DDCs & DUCs would be arranged as shown below:



DDCs are internal DLMAN structures used for overall device control, DUCs facilitate control at the sub-unit level. Pointers to DUCs are used as unit (and hence device) identifiers in the DLMAN APIs.

For each device Windows calls the display driver at PnPInit time with the devices primary DEVNODE. The display driver will in turn call **DLMANInitDevice()** which creates a single DDC and a DUC for each unit on that device and returns a pointer to the DDC. A DUC is created for all units which can be supported by the particular device configuration irrespective of whether an attached monitor is sensed.

Units are usable as independent surfaces in Windows once a DEVNODE to DUC association has been established through a call to **DLMANAssocOSDevToVLUnit().** This normally occurs when Windows wants to establish a mode on that unit (through display drivers **Enable()**).

Windows knows how many units are available on a device by calling the display driver **GetUnitInfo()** function. This reports either 1 or 2 units to Windows for each PMX device in the system, dependent upon the users choice of whether to use the TV in independent mode or not (PMX1-LC will never report 2 units since it cannot support independent displays).

A DDC has a one to one association with a Kernel Manager Display Control Block (DCB).

In order to establish it's first mode, the display driver calls **DLMANCaps()** for the default DUC, initialises monitor data appropriately and calls **DLMANDisplayMode()**.

The format of the DMDEVCTL (DDC) is private to DLMAN and exists here for reference:-

```
typedef struct tagDMDEVCTL
{
      DWORD dwTVEncoderPresent;     // The board possesses a TV encoder
      DWORD dwTVEncoderRev; // Revision of the encoder
      MEMSTATAL   sCursorData;      // Cursor data store in framebuffer
      DWORD       dwOSDev;          // Primary Devnode for device
      PDEVICE_CTLBLK    psDCB;      // KM Device Control Blk pointer
      KMCARDINFO        sKMInfo;    // KM Device Info
      DWORD             dwUnits;    // Units supported by Device
      DWORD             dwMirror;   // Simultaneous mode?
      DWORD             dwMaxMemBW; // Max memory bandwidth for device
      PVOID             pvDOSBoxMem;// Ptr to DOS Box data area
      POVERLAY          psOvl;      // Base of all overlays
      PBUFFER           psBuf;      // base of all buffers for overlays
      struct tagDMUNITCTL     *psUnit;    // Ptr to first unit on device
      struct tagDMDEVCTL      *psNext;    // Link to next device
      BYTE              bActiveCursor;    //
      DWORD             dwResetFlip;    // Mask to say what we can flip to
      MEMMOVE sBlankingArea;        // pointer to blank memory for blanking
region
}
DMDEVCTL;
```

The format of the DMUNITCTL (DUC) is private to DLMAN and exists here for reference:-

```
typedef struct tagDMUNITCTL
{
      DWORD                   dwUnit;     // The unit (CRT, TV, LCD)
      DWORD                   dwOSDev;    // OS Cookie (DEVNODE)
      UNITCAPS                sCaps;      // Base Caps
      UNITCAPS                sMCaps;     // Caps     for the current mode
      UNITXCAPS               sXCaps;     // Extended Caps
      UNITSENSE               sSense;     // Sensed data
      UNITREGS                sCurRegs;   // Current register set
      UNITREGS                sDefRegs;   // Default register set
```

```
    PSURFACE                psSCtl;      // surface ctl struct
    DWORD                   dwTop;  /* current max Z in depth units */
    DWORD                   dwBot;  /* current min Z in depth units */
    DWORD                   dwMaxDotClk; // Max pixel clock supported
    DWORD                   dwCurMemBW; // Memory bandwidth used by
                                        //current mode
    PRTDISPLAYIF            psRTComms;
    PDLCTL                  apsDLTab[2][4][4][4][4][4];
    CMDIF                   sAllocIF;
    CODE_DETAILS            sCodeDetails;
    DLCMDIF                 sDLCmdIF;
    struct tagDMDEVCTL      *psParent;  // Ptr to parent DDC
    struct tagDMUNITCTL     *psNext;    // Ptr to next unit on device
    BYTE                    bActive;    // Unit active?
    BYTE                    bActiveDLTab;
    BYTE                    bNumSurf;
    BYTE                    bUsed;// Unit used by client in this session?
} DMUNITCTL;
```

Units can act either independently or in simultaneous mode. This is decided at **DLMANDisplayMode()** time when a unit can be specified to operate in simultaneous mode, as long as it's resolution, bit depth and refresh rate are common with the mode on the primary unit (the unit which the display driver associated the primary DEVNODE with).

The behaviour of a number of APIs are affected by simultaneous mode. Those APIs which affect cursors and overlays act simultaneously on both outputs when in simultaneous mode and the primary DUC must be specified in the API.

The APIs affecting screen display attributes (e.g. Brightness, Gamma) and screen size/position operate independently on the outputs despite simultaneous mode operation and the desired DUC must be specified in the API. This behaviour is not completely supported in PMX1-LC which is limited to same screen positioning and Gamma attributes.

All APIs are common between all OS-versions of DLMAN. However, the manner in which the service table defining the API entry points is returned to a client is OS-specific. DLMAN provides direct access to its services only to other VxDs (Windows 9x) or Kernel Mode drivers (NT). The generic **DLMANGetServiceTable()** API is accessed via a Windows VxD service call or via the NT DIOCTL interface and is not therefore a member of the service table exported by DLMAN for access to all other APIs. The initialisation portion of DLMAN reflects the OS-dependent access to **DLMANGetServiceTable()**.

In the following specification, the dwFlags parameter, where present in an API, with the exception of **DLMANDpms(),** facilitates default or new data setting or getting. The dwFlags

parameter takes the values 0 or 1 for a set or get operation, respectively and either value may be OR'd with 0x80000000 to specify a get or set of the default dataset. In addition, and uniquely for use with **DLMANDisplayMode()**, the dwFlags parameter can take a validate flag indicating that the API should compute all mode registers from the input mode specification but should not set the mode. The following defines are exported for this purpose:-

```
#define DL_SET          0
#define DL_GET          1
#define DL_GETSETMASK   0x00000001
#define DL_NEW          0
#define DL_DEF          0x80000000
#define DL_NEWDEFMASK   0x80000000

#define DL_VALID  (0x40000000 | DL_SET)

#define DL_SET_DEF  (DL_DEF | DL_SET)
#define DL_GET_DEF  (DL_DEF | DL_GET)
```

DLMAN can return the following errors:-

```
/*****************************************************************************
 API error and warning flags.
*****************************************************************************/
#define DL_OK                                0x00000000
#define DL_NO_DEV                            0x00000001
#define DL_NO_UNIT                           0x00000002
#define DL_ERROR_INVALID_API                 0x00000003
#define DL_ERROR_INVALID_PARAMS              0x00000004
#define DL_ERROR_NOTSUPPORTED                0x00000005
#define DL_CANT_ALLOCATE_HOST_MEMORY         0x00000006
#define DL_CANT_ALLOCATE_FB_MEMORY           0x00000007

#define DL_VALUE_OUT_OF_RANGE                0x00000008
#define DL_NO_ROOM_FOR_IBC                   0x00000009
#define DL_DOTCLK_NOT_POSSIBLE               0x0000000A
#define DL_DAC_THREAD_NOT_ACTIVE             0x0000000B
#define DL_TV_THREAD_NOT_ACTIVE              0x0000000C
#define DL_INVALID_DLIST_ID                  0x0000000D
#define DL_NOT_PRIMARY_DEV                   0x0000000E
#define DL_UNKNOWN_FRAGTYPE                  0x0000000F

#define DL_CANT_LOAD_DACFRAG                 0x00000010
#define DL_CANT_LOAD_TVFRAG                  0x00000011
#define DL_CANT_ALLOC_CURSOR_STORE           0x00000012
#define DL_CANT_GAMMA_AT_COLOURDEPTH         0x00000013
#define DL_I2C_BUS_ERROR                     0x00000014
#define DL_TIMEOUT                           0x00000015
```

3.1. Get Service Table

```
DWORD DLMANGetServiceTable(PVOID pvIn, PVOID pvOut)
```

```
PURPOSE         A DIOCTL interface or VxD service function which returns the
                service table for the Kernel Mode driver or VxD DLMAN.

PARAMETERS      pvIn        - pointer to an IODLSERVICE structure
                pvOut       - unused

RETURNS         DL_OK
```

Under Win9x, DLMAN instantiates a system-specific inter-VxD service table consisting of the **DLMANGetServiceTable()** function pointer. Clients retrieve the generic DLMAN service table through the VxDCall macro which encodes the Microsoft-assigned DLMAN VxD identifying number within the exported **DLMANGetServiceTable** symbol and takes, as parameters, the pvIn and pvOut parameters of the **DLMANGetServiceTable()** function itself. Under NT, The **DriverEntry()** routine in DLMAN creates a kernel mode driver function table entry for a Device IO Control entry point, through which the client can call to the **DLMANGetServiceTable()** API, as a DIOCTL sub-function.

Catering for both OS methods of client access in a generic fashion, the IODLSERVICE structure is defined below, along with the DLMAN_SERVICE_TABLE structure itself:-

```
/****************************************************************************
 Service Table details
****************************************************************************/

typedef struct tagDLMAN_SERVICE_TABLE
{
        DWORD dwSize;
        DWORD (__stdcall *DLMANInitDevice)(DWORD dwOSDev, DWORD dwFlags, PDMDEVCTL *ppsDDC);
        DWORD (__stdcall *DLMANDeInitDevice)(PDMDEVCTL psDDC);
        DWORD (__stdcall *DLMANAssocOSDevToVLUnit)(DWORD dwOSDev, PDMDEVCTL psDDC, DWORD
dwUnitType, PDMUNITCTL *ppsDUC);
        DWORD (__stdcall *DLMANMapOSDevToVLDev)(DWORD dwOSDev, PDMDEVCTL *ppsDDC);
        DWORD (__stdcall *DLMANMapOSDevToVLUnit)(DWORD dwOSDev, PDMUNITCTL *ppsDUC);
        DWORD (__stdcall *DLMANEnumerateDevices)(DWORD dwEnum, PDWORD pdwOSDev, PDMDEVCTL
*ppsDDC);
        DWORD (__stdcall *DLMANEnumerateUnits)(DWORD dwEnum, DWORD dwOSDev, PDMUNITCTL
*ppsDUC);
        DWORD (__stdcall *DLMANCaps)(PDMUNITCTL psDUC, DWORD dwModal, PUNITCAPS psCaps);
        DWORD (__stdcall *DLMANSense)(PDMUNITCTL psDUC, PUNITSENSE psSense);
        DWORD (__stdcall *DLMANDisplayMode)(PDMUNITCTL psDUC, DWORD dwFlags, PDLMODE psMode);
        DWORD (__stdcall *DLMANShowCursor)(PDMUNITCTL psDUC, DWORD dwShow);
        DWORD (__stdcall *DLMANSetCursorPos)(PDMUNITCTL psDUC, DWORD dwX, DWORD dwY);
        DWORD (__stdcall *DLMANSetCursorShape)(PDMUNITCTL psDUC, PCSHAPE psShape);
        DWORD (__stdcall *DLMANPanningRegs)(PDMUNITCTL psDUC, DWORD dwFlags, PPANNINGREGS
psRegs);
        DWORD (__stdcall *DLMANTimingRegs)(PDMUNITCTL psDUC, DWORD dwFlags, PTIMINGREGS
psRegs);
        DWORD (__stdcall *DLMANBCSHRegs)(PDMUNITCTL psDUC, DWORD dwFlags, PBCSHREGS psRegs);
        DWORD (__stdcall *DLMANPicture)(PDMUNITCTL psDDC, DWORD dwFlags, PPQUAL psPQual);
        DWORD (__stdcall *DLMANDpms)(PDMUNITCTL psDUC, DWORD dwFlags, PDWORD pdwState);
        DWORD (__stdcall *DLMANCKey)(PDMUNITCTL psDUC, DWORD dwFlags, PCKREGS psRegs);
        DWORD (__stdcall *DLMANPalette)(PDMUNITCTL psDUC, DWORD dwFlags, DWORD dwIndex, DWORD
dwCount, PBYTE pbPal);
        DWORD (__stdcall *DLMANGamma)(PDMUNITCTL psDUC, DWORD dwFlags, PLONG plGamma);
        DWORD (__stdcall *DLMANCreateDList)(PDMUNITCTL psDUC, PCREATEDL psCreate);
        DWORD (__stdcall *DLMANDestroyDList)(PDMUNITCTL psDUC, DWORD dwDLID);
        DWORD (__stdcall *DLMANGetDListID)(PDMUNITCTL psDUC, PDWORD pdwDLID);
        DWORD (__stdcall *DLMANAttachDList)(PDMUNITCTL psDUC, DWORD dwDLID);
        DWORD (__stdcall *DLMANDetachDList)(PDMUNITCTL psDUC, DWORD dwDLID);
        DWORD (__stdcall *DLMANCreateOverlay)(PDMUNITCTL psDUC, PCREATEOVL psCreate);
        DWORD (__stdcall *DLMANDestroyOverlay)(PDMUNITCTL psDUC, DWORD dwOID);
        DWORD (__stdcall *DLMANSetOverlayAttributes)(PDMUNITCTL psDUC, PATTROVL psAttr);
        DWORD (__stdcall *DLMANAttachOverlay)(PDMUNITCTL psDUC, DWORD dwDLID, DWORD dwOID);
```

```
        DWORD (__stdcall *DLMANDetachOverlay)(PDMUNITCTL psDUC, DWORD dwDLID, DWORD dwOID);
        DWORD (__stdcall *DLMANCreateBuffer)(PDMUNITCTL psDUC, PCREATEBUF psCreate);
        DWORD (__stdcall *DLMANDestroyBuffer)(PDMUNITCTL psDUC, DWORD dwBID);
        DWORD (__stdcall *DLMANAttachBuffer)(PDMUNITCTL psDUC, DWORD dwOID, DWORD dwBID);
        DWORD (__stdcall *DLMANDetachBuffer)(PDMUNITCTL psDUC, DWORD dwOID, DWORD dwBID);
        DWORD (__stdcall *DLMANStartI2C)(PDMUNITCTL psDUC);
        DWORD (__stdcall *DLMANStopI2C)(PDMUNITCTL psDUC);
        DWORD (__stdcall *DLMANReadI2C)(PDMUNITCTL psDUC, BYTE bSlave, DWORD dwCount, PBYTE
pbBuf);
        DWORD (__stdcall *DLMANWriteI2C)(PDMUNITCTL psDUC, BYTE bSlave, DWORD dwCount, PBYTE
pbBuf);
        DWORD (__stdcall *DLMANResetI2C)(PDMUNITCTL psDUC);
}
DLMAN_SERVICE_TABLE, *PDLMAN_SERVICE_TABLE;
```

Any **DLMANDummy**() entries in the service table reflect APIs that are no longer in existance. **DLMANDummy()** returns DL_ERROR_INVALID_API, only.

```
typedef struct tagIO_HEADER
{
        DWORD dwPktSize;
        DWORD dwError;
        DWORD dwVLDev;
}
IO_HEADER, *PIO_HEADER;

typedef struct tagDLSERVICE
{
        DWORD                               dwNumServices;
        struct tagDLMAN_SERVICE_TABLE *psServiceTable;
}
DLSERVICE, *PDLSERVICE;

typedef struct tagIODLSERVICE
{
        IO_HEADER    sIOH;
        DLSERVICE    sServ;
}
IODLSERVICE, *PIODLSERVICE;
```

3.2. Initialisation

```
DWORD DLMANInitDevice(DWORD dwOSDev, DWORD dwFlags, PDMDEVCTL *ppsDDC)

PURPOSE         Initialises DLMAN on a per device basis, including the
                creation of DUCs.

PARAMETERS      dwOSDev    - Windows primary DEVNODE device identifier
                dwFlags    - Flags word. Currently reserved.
                ppsDDC     - pointer through which to return the created psDDC


RETURNS         DL_OK, if successful else error code.
```

Initialises DLMAN. DLMAN creates a DDC and then a DUC for each unit supported on the device.

This function needs to be called for each primary DEVNODE in the system.

```
DWORD __stdcall DLMANEnableDevice(PDMDEVCTL psDDC)
Purpose         Loads up threads
Parameters      psDDC       - Pointer to DDC which is being enabled.
Returns         DL_OK if ok else error code
```

This is responsible for loading up the threads , this should be called whenever the DAC thread code has been unloaded, such as when we have gone into a VGA mode or Low Power Mode.

3.3. De-initialisation

```
DWORD DLMANDeInitDevice(PDMDEVCTL psDDC)
PURPOSE         DeInitialises DLMAN devices, deallocates memory, etc
PARAMETERS      psDDC       - device to de-initialise

RETURNS         DL_OK, if successful else error code.
```

De-initialises DLMAN. This function needs to be called for each primary DEVNODE in the system at session end.

3.4. Enumerate Devices

```
DWORD DLMANEnumerateDevices(DWORD dwEnum, PDWORD pdwOSDev,
                                                    PDMDEVCTL *ppsDDC)
PURPOSE         Returns a DDC and the OS-specific device identifier
                associated with the requested enumeration level.
PARAMETERS      dwEnum      - Requested device enumeration level
                dwOSDev     - Windows primary DEVNODE device identifier
                psDDC       - pointer to a DMDEVCTL structure
RETURNS         DL_OK, if successful else error code.
```

This API is used to determine what devices are supported in DLMAN. It is called first with the dwEnum parameter equal to zero. On a non-error return with a non-zero value for the DDC, the user may increment dwEnum and reissue the command to enumerate the next device. Enumeration may continue until an error condition is returned. The DDC will be zero in this case. Units supported by the device are enumerated by using **DLMANEnumerateUnits()**

3.5. Enumerate Units

```
DWORD DLMANEnumerateUnits(DWORD dwEnum, DWORD dwOSDev, PDMUNITCTL *ppsDUC)
```

```
PURPOSE         Returns a DUC on the given device (identified by dwOSDev)
                associated with the requested enumeration level.

PARAMETERS      dwEnum      - Requested unit enumeration level
                dwOSDev     - OS device identifying cookie
                ppsDUC      - pointer through which to return a DUC ptr

RETURNS         DL_OK, if successful else error code.
```

This API is used to determine what display units are supported by DLMAN on the indicated device. It is called first with the dwEnum parameter equal to zero. On a non-error return with a non-zero value for the DUC, the user may increment dwEnum and reissue the command to enumerate the next unit. Enumeration may continue until an error condition is returned. The DUC will be zero in this case. Further information on the DUCs can be gleaned from the **DLMANCaps()** and **DLMANSense()** APIs.

3.6. Associate OS-Device To VL-Unit

```
DWORD DLMANAssocOSDevToVLUnit(DWORD dwOSDev, PDMDEVCTL psDDC,
                                    DWORD dwType, PDMUNITCTL *ppsDUC)

PURPOSE         Associates a DUC with a Devnode. This allows use of a DUC as
                a Windows surface.

PARAMETERS      dwOSDev     - Windows DEVNODE device identifier
                psDDC       - DDC associated with primary DEVNODE
                dwType      - Sub unit type indicated by DEVNODE
                ppsDUC      - pointer through which to return a DUC ptr

RETURNS         DL_OK, if successful else error code.
```

dwOSDev may be either a primary or subsidiary DEVNODE. The psDDC indicated the base device for which the DEVNODE should be associated with a DUC.

The dwType specifies which type (CRT, LCD, TV) of unit the DEVNODE represents (DLMAN will choose if dwType is 0) and the function returns the fully initialised DUC through ppsDUC. The dwType parameter can be one of:-

```
#define     DL_CRT_UNIT         0x01
#define     DL_LCD_UNIT         0x02
#define     DL_TV_UNIT          0x04
```

It is through this function that DUCs are first made available to the caller.

This call must be made before **DLMANMapOSDevToVLUnit()** and **DLMANEnumerateUnits()** can succeed.

3.7. Map OS-Device To VL-Device

```
DWORD DLMANMapOSDevToVLDev(DWORD dwOSDev, PDMDEVCTL *ppsDDC)
```

```
PURPOSE         Translates from an OS cookie that identifies the hardware
                device to a DMDEVCTL Block in DLMAN.

PARAMETERS      dwOSDev    - Windows DEVNODE device identifier
                ppsDDC     - pointer through which to return a psDDC

RETURNS         DL_OK, if successful else error code.
```

This call takes an OS Cookie (a DEVNODE in Windows 9x) and translates it into a DDC which is unique for a specific DLMAN-managed device. This DDC can then be used in DLMAN API calls that required it. It is used by any client who needs to retrieve a DDC for use with DLMAN.

3.8. Map OS-Device To VL-Unit

```
DWORD DLMANMapOSDevToVLUnit(DWORD dwOSDev, PDMUNITCTL *ppsDUC);

PURPOSE         Translates from an OS cookie that identifies the hardware
                device to a DMUNITCTL Block in DLMAN.

PARAMETERS      dwOSDev    - Windows DEVNODE device identifier
                ppsDUC     - pointer through which to return a psDUC

RETURNS         DL_OK, if successful else error code.
```

This call takes an OS device-identifying cookie (a DEVNODE in Windows 9x) and translates it into a DUC which is unique for a specific unit. This DUC is then used in the other DLMAN API calls. It is used by any client who needs to retrieve a DUC for use with DLMAN.

3.9. Set/Get Display Mode

```
DWORD DLMANDisplayMode(PDMUNITCTL psDUC, DWORD dwFlags, PDLMODE psMode)

PURPOSE         Creates the display list to reflect the requested mode.

PARAMETERS      psDUC      - pointer to units DMUNITCTL structure.
                dwFlags    - flags word indicating set/get (default/current)
                psMode     - pointer to DLMODE structure.

RETURNS         DL_OK, if successful else error code.
```

This API establishes a new screen mode by creating a display list and setting the relevant pixel pipeline up in an appropriate manner. It is used for establishing or validating modes on the CRT, TV or LCD outputs. It can also be used to disable and subsequently re-enable the output to a specific device. Enabling (or disabling) the display unit's output is logically separate to establishing the display mode and is effected with or without changing the current mode via the DM_NOCHANGE and DM_ONOFF flags (psMode->dwFlags). The DM_NOCHANGE, DM_ONOFF and DM_MIRROR flags are only actioned if DL_SET or DL_SET_DEF (dwFlags) is specified.

It is possible to request that a device's output is a copy of another output (e.g. the TV output copies the CRT output – "simultaneous TV"). In such a case all subsequent calls which affect the display list of the master output (CRT) are reflected on the slave output. In this case it is

the callers responsibility to establish the master output mode first, followed by the slave output mode – both modes having the same resolution, bit depth and refresh rates and both modes being supported on both outputs. DLMAN will check for and fail illegal conditions (e.g. attempting to set up a 1280x1024 mode on the TV).

On PMX1-LC simultaneous TV is the only mode in which output to both CRT and TV are supported at the same time.

DLMANCaps() should be called following the establishment of a mode to check on the mode-specific capabilities of each unit.

```
/*****************************************************************************
 Definitions for dwFlags member of DLMODE in DLMANDisplayMode() API
*****************************************************************************/
#define DM_ONOFF        0x00000001  /* enable/disable device */
#define DM_MIRROR       0x00000002  /* mirror primary device */
#define DM_NOCHANGE     0x00000004  /* do not change mode */

/*****************************************************************************
 Definitions for dwFlags member of CRTMODE in DLMANDisplayMode() API
*****************************************************************************/
#define CRT_ILACE       0x00000001  /* interlaced mode */
#define CRT_LOREZ       0x00000002  /* pix and line doubled (lo-rez) mode*/
#define CRT_VESAVGA     0x00000004  /* legacy VGA calc else GTF method */
#define CRT_HSVS        0x00000008  /* use supplied HS/VS polarities */
#define CRT_NEAREST     0x00000010  /* set/get nearest mode */
#define CRT_FRAME       0x00000000  /* use frame rate for mode calc */
#define CRT_LINE        0x00010000  /* use line rate for mode calc */
#define CRT_CLOCK       0x00020000  /* use clock rate for mode calc */

#define CRT_MODECTL_MASK      0x0000001F  /* ilace, lorez, nearest, etc. */
#define CRT_METHOD_MASK       0x00030000  /* use Hz, KHz*1000 or MHz*1000*/

/*****************************************************************************
 Definitions for dwSyncs member of CRTMODE, LCDMODE in DLMANDisplayMode()
*****************************************************************************/
#define CRT_HSN         0x00000000  /* HS pulse polarity is -ve */
#define CRT_HSP         0x00000001  /* HS pulse polarity is +ve */
#define CRT_VSN         0x00000000  /* VS pulse polarity is -ve */
#define CRT_VSP         0x00000002  /* VS pulse polarity is +ve */

/*****************************************************************************
 Definitions for dwControl member of TVMODE in DLMANDisplayMode() API
*****************************************************************************/
#define TV_SCART169     0x00000002  /* use scart with 16:9 av */
#define TV_SCART43      0x00000003  /* use scart with 4:3 av */

/*****************************************************************************
 Definitions for dwSystem member of TVMODE in DLMANDisplayMode API
*****************************************************************************/
#define TV_UNDERSCAN    0x80000000  /* or overscan */
#define TV_SQPIX        0x40000000  /* or CCIR601 */
#define TV_PAL          0x00000000  /* PAL-BDGHI */
#define TV_PAL_M        0x00000001  /* PAL-M */
```

```
#define TV_PAL_N          0x00000002  /* PAL-N */
#define TV_PAL_N_DASH     0x00000003  /* PAL-N' */
#define TV_NTSC           0x00000004  /* NTSC-M */
#define TV_NTSC_J         0x00000005  /* NTSC-J */

/*****************************************************************************
 Definitions for dwFormat member of TVMODE in DLMANDisplayMode API
*****************************************************************************/
#define TV_CVBS           0x00000000  /* CVBS on phono */
#define TV_CVBS_Y         0x00000001  /* CVBS on luma pin of mini-din */
#define TV_CVBS_C         0x00000002  /* CVBS on chroma pin of mini-din */
#define TV_SVIDEO         0x00000003  /* SVIDEO on min-din */
#define TV_RGB            0x00000004  /* RGB on 25-way D-Type */
#define TV_SOG            0x00000005  /* RGB sync-on-green on 25-way D-Type*/

/*****************************************************************************
 Definitions for dwFlags member of LCDMODE in DLMANDisplayMode API
*****************************************************************************/
#define LCD_ZOOM          0x00000001  /* zoom */
#define LCD_PORT          0x00000002  /* portrait else landscape */
#define LCD_HSVS          0x00000004  /* use supplied HS/VS polarities */

/*****************************************************************************
 DLMANDisplayMode() structs
*****************************************************************************/
typedef struct tagCRTMODE
{
      DWORD dwFlags;            /* GTF Method, Lo-Rez, Interlace, Force
                                Syncs, Nearest */
      DWORD dwVtFreq;           /* Vertical Freq in Hz */
      DWORD dwHzFreq;           /* Horizontal Freq also in Hz */
      DWORD dwClkFreq;          /* Clock Frequency in KHz */
      DWORD dwHBlankRatio;      /* percentage HBlank (legacy modes only) */
      DWORD dwVBlankRatio;      /* percentage VBlank (legacy modes only) */
      DWORD dwSyncPol;          /* HS and VS polarity override */
      PWORD awRefreshes;        /* 0-term'd list of VtFreqs if CRT_NEAREST */
}
CRTMODE, *PCRTMODE;

typedef struct tagLCDMODE
{
      DWORD dwFlags;            /* Force Syncs, Zoom, Landscape/Portrait */
      DWORD dwVtFreq;           /* Vertical Freq in Hz */
      DWORD dwHBlankRatio;      /* percentage HBlank (legacy modes only) */
      DWORD dwVBlankRatio;      /* percentage VBlank (legacy modes only) */
      DWORD dwSyncPol;          /* HS and VS polarity override */
}
LCDMODE, *PLCDMODE;

typedef struct tagTVMODE
{
      DWORD dwFlags;            /* Reserved */
      DWORD dwSystem;           /* Under/OverScan, CCIR601/SQ, PAL, NTSC */
      DWORD dwFormat;           /* SOG, RGB, CVBS, SVIDEO */
      DWORD dwLumaDelay;        /* Luma Delay (4 bit value, usually 7) */
```

```
}
TVMODE, *PTVMODE;

typedef struct tagDLMODE
{
      DWORD dwFlags;     /* On/off, mirror */
      DWORD dwXExt;      /* Display surface x extent (pixels) */
      DWORD dwYExt;      /* Display surface y extent (lines) */
      DWORD dwStride;    /* Line stride of surface in pixels */
      DWORD dwBPP;       /* Bits per pixel - 8, 15, 16, 24, 32 */
      DWORD dwPhysAddr; /* Address of primary display surface */
      union
      {
            CRTMODE sCRTMode; /* CRT specific mode parameters */
            TVMODE  sTVMode;  /* TV  specific mode parameters */
            LCDMODE sLCDMode; /* LCD Specific mode parameters */
      };
}
DLMODE, *PDLMODE;
```

3.10. Show Cursor

```
DWORD DLMANShowCursor(PDMUNITCTL psDUC, DWORD dwShow)

PURPOSE        Hide or Show hardware cursor.

PARAMETERS     psDUC     - pointer to units DMUNITCTL structure.
               dwShow    - Required state of cursor visibility.
                           0 hide cursor else make it visible.

RETURNS        DL_OK, if successful else error code.
```

This API is called by the 2D display driver in response to GDI's calling of the driver's **SetCursor()** entry point, in the establishment of monochrome hardware cursor visibility.

3.11. Set Cursor Position

```
DWORD DLMANSetCursorPos (PDMUNITCTL psDUC, DWORD dwX, DWORD dwY)

PURPOSE        Set new pixel X/Y position for hardware cursor.

PARAMETERS     psDUC     - pointer to units DMUNITCTL structure.
               dwX, dwY  - X and Y pixel positions of the cursor.

RETURNS        DL_OK, if successful else error code.
```

This API is called by the 2D display driver in response to GDI's calling of the driver's **MoveCursor()** entry point, in the establishment of monochrome hardware cursor position.

3.12. Set Cursor Shape

```
DWORD DLMANSetCursorShape(PDMUNITCTL psDUC, PCSHAPE psShape)
```

```
PURPOSE         Set new hardware cursor shape.

PARAMETERS      psDUC      - pointer to units DMUNITCTL structure
                psShape    - pointer to CSHAPE structure

RETURNS         DL_OK, if successful else error code.
```

This API is called by the 2D display driver in response to GDI's calling of the driver's **SetCursor()** entry point, in the establishment of monochrome hardware cursor appearance. The CSHAPE structure, detailed below is identical with that used by **SetCursor()** and can be passed straight down by the driver.

```
typedef struct tagCSHAPE
{
      WORD  wXHot;              /* pixel offset to X hot spot in cursor */
      WORD  wYHot;              /* line offset to Y hot spot in cursor */
      WORD  wXExt;              /* X-directed pixel size of cursor (32,64) */
      WORD  wYExt;              /* Y-directed pixel size of cursor (32,64) */
      WORD  wStride;            /* XOR data line stride */
      BYTE  bPlanes;            /* Always 1 */
      BYTE  bBPP;               /* 01h = mono, 08h = 8bpp, etc. */
      DWORD adwShape[1];        /* AND then XOR mask data */
}
CSHAPE, *PCSHAPE;
```

The adwShape[] member of CSHAPE contains the NxN (N=32 or 64) pixels of AND mask data followed by the NxN bits of XOR mask data. For a mono cursor, currently the only supported hardware cursor type, adwShape[] specifies 2NxN bits (=1024 or 256 bytes) of data. For colour cursors, the AND mask contains pixel data of size equivalent to the bits-per-pixel specified in the bBPP member, while the XOR mask is always a mono bitmap mask.

3.13. Set/Get Panning Registers

```
DWORD DLMANPanningRegs(PDMUNITCTL psDUC, DWORD dwFlags, PANNINGREGS psRegs)

PURPOSE         Alter panning registers. This allows for virtual desktops to
                be supported.

PARAMETERS      psDUC      - pointer to units DMUNITCTL structure.
                psRegs     - pointer to Panning Registers structure.
                dwFlags    - Set registers or Get current registers or
                             get default registers

RETURNS         DL_OK, if successful else error code.
```

The dwMask member of the PANNINGREGS structure establishes a bit position to register correspondence as below:-

Bit	Register	Description
b0	SCRSTOFF	Screen start offset register.
b1	LOGSCRX	Logical screen extent register.
b2	VDPAN	Virtual desktop viewport panning register.

The following defines are exported for this purpose:-

```
/*****************************************************************************
 Bit position definitions for dwMask member of PANNINGREGS struct
*****************************************************************************/
#define SCRSTOFF   0x00000001        /* Screen Start Offset Reg */
#define LOGSCRX    0x00000002        /* Logical Screen Extent Reg */
#define VDPAN      0x00000004        /* Virtual Desktop Panning Reg */
```

3.14. Set/Get Timing Registers

```
DWORD DLMANTimingRegs(PDMUNITCTL psDUC, DWORD dwFlags, PTIMINGREGS psRegs)

PURPOSE        Set or Get the timing registers for the current mode to
               allow an application to resize / position the viewable
               output with respect to the monitor.

PARAMETERS     psDUC      - pointer to units DMUNITCTL structure.
               dwFlags    - Set timing registers or Get current registers or
               Get default mode registers.
               psRegs     - pointer to TIMINGREGS structure.

RETURNS        DL_OK, if successful else error code.
```

The dwMask member of the TIMINGREGS structure is a bit mask indicating which pseudo CRTC register(s) are affected by the API. The bit position to register correspondence is as follows:-

Bit	Register	Description
b0	HDISPEND	Pixel position of start of horizontal blanking. e.g. in a screen of 1024 visible pixels in a line, HDISPEND would normally have a value of 1024.
b1	HSYNCST	Pixel position of start of horizontal sync.
b2	HSYNCEND	Pixel position of end of horizontal sync.
b3	HTOTAL	Total number of pixels (active plus blanking) in a line.
b4	VDISPEND	Line position of start of vertical blanking. E.g in a screen of 768 visible lines, VDISPEND would normally have a value of 768.
b5	VSYNCST	Line position of start of vertical sync.
b6	VSYNCEND	Line position of end of vertical sync.
b7	VTOTAL	Total number of lines (active plus blanking) in a screen.

The following defines are exported for this purpose:-

```
/*****************************************************************************
 Bit position definitions for dwMask member of TIMINGREGS struct
*****************************************************************************/
#define HDISPEND   0x00000001
#define HSYNCST    0x00000002
#define HSYNCEND   0x00000004
#define HTOTAL     0x00000008
#define VDISPEND   0x00000010
#define VSYNCST    0x00000020
```

```
#define VSYNCEND   0x00000040
#define VTOTAL     0x00000080
```

3.15. Get/Set Brightness, Contrast, Saturation and Hue

```
DWORD DLMANBCSHRegs (PDMUNITCTL psDUC, DWORD dwFlags, PBCSHREGS psRegs)

PURPOSE        Set or Get the Brightness, Contrast, Saturation and/or Hue
               for the specified unit.

PARAMETERS     psDUC      - pointer to units DMUNITCTL structure.
               dwFlags    - Set or Get BCSH settings.
               psBCS      - pointer to BCSHREGS structure.

RETURNS        DL_OK, if successful else error code.
```

The dwMask member of the BCSHREGS structure is a bit mask indicating which of B, C, S and/or H are affected by the API. The bit position to register correspondence is as follows:-

b0 BRI
b1 CON
b2 SAT
b3 HUE

The following defines are exported for this purpose:-

```
/*****************************************************************************
 Bit position definitions for dwMask member of BCSHREGS struct
*****************************************************************************/
#define BRI              0x00000001
#define CON              0x00000002
#define SAT              0x00000004
#define HUE              0x00000008
```

3.16. Get/Set DPMS

```
DWORD DLMANDpms(PDMUNITCTL psDUC, DWORD dwFlags, PDWORD pdwState)

PURPOSE        To return information on the DPMS capabilities or to set the
               DPMS state of the given unit.

PARAMETERS     psDUC      - pointer to units DMUNITCTL structure.
               dwFlags    - Get/Set Capabilities/State
               pdwState   - Pointer to data to set/get.

RETURNS        DL_OK, if successful else error code.
```

This function sets or gets the Display Power Management System (DPMS) state and/or capabilities of the indicated display unit. The dwFlags parameter is created from an OR operation of 0 or 1 (set or get) with 0 or 0x8000000 (state or capabilities specifier). Capabilities are returned as an OR'd value of the DPMS_STATEx defines indicating which DPMS states are supported.

```
/*****************************************************************************
 DLMANDpms() API dwFlags parameter defines
*****************************************************************************/
#define DPMS_SET        0x00000000
#define DPMS_GET        0x00000001
#define DPMS_CAPS       0x80000000
#define DPMS_STATE      0x00000000

/*****************************************************************************
 DLMANDpms() API dwState parameter defines
*****************************************************************************/
#define DPMS_STATE0     0x00000001
#define DPMS_STATE1     0x00000002
#define DPMS_STATE2     0x00000004
#define DPMS_STATE3     0x00000008
```

3.17. Get Display Device Capabilities

```
DWORD DLMANCaps(PDMUNITCTL psDUC, DWORD dwModal, PUNITCAPS psCaps)

PURPOSE       To return information on the capabilities of the required
              unit.

PARAMETERS    psDUC     - pointer to units DMUNITCTL structure.
              dwModal   - Return mode-specific caps
              psCaps    - pointer to Caps structure.

RETURNS       DL_OK, if successful else error code.
```

This function returns the capabilities of the specified unit together with overall information on the device the unit is attached to. The dwModal parameter allows the return of capabilities that are valid for the display mode currently in effect.

```
/*****************************************************************************
 DLMANDevCaps() and DLMANDevSense() capabilities defines
*****************************************************************************/
/* CRT, TV and LCD screen */
#define DLCAPS_ONOFF            0x00000001  /* Can enable/disable device */
#define DLCAPS_GAMMA            0x00000002  /* Can do gamma correction */
#define DLCAPS_BRIGHTNESS       0x00000004  /* Can modify bri */
#define DLCAPS_CONTRAST         0x00000008  /* Can modify con */
#define DLCAPS_SATURATION       0x00000010  /* Can modify sat */
#define DLCAPS_HUE              0x00000020  /* Can modify hue */
#define DLCAPS_SHARPNESS        0x00000040  /* Can modify picture quality */
#define DLCAPS_FLICKER          0x00000080  /* Can flicker filter */
#define DLCAPS_X_INTERPOL       0x00000100  /* Can interpolate in X */
#define DLCAPS_Y_INTERPOL       0x00000200  /* Can interpolate in Y */
#define DLCAPS_PAN              0x00000400  /* Can pan port around desktop */
#define DLCAPS_POSITION         0x00000800  /* Can pos surface wrt monitor */
#define DLCAPS_SIZE             0x00001000  /* Can size surface wrt monitor*/
#define DLCAPS_MIRROR           0x00002000  /* Can mirror */
#define DLCAPS_DPMS             0x00004000  /* Can do DPMS */

/* CRT, TV, and LCD window */
```

```
#define DLCAPS_OVERLAY          0x00000001  /* Can do overlay */
#define DLCAPS_COLOURKEY        0x00000002  /* Can do colour keying */
#define DLCAPS_CHROMAKEY        0x00000004  /* Can do chroma keying */
#define DLCAPS_BOB              0x00000008  /* Can do bobbing */
#define DLCAPS_WEAVE            0x00000010  /* Can do weaving */
#define DLCAPS_FIELD            0x00000020  /* Can do field mode */
#define DLCAPS_FRAME            0x00000040  /* Can do frame mode */

/* CRT, TV, and LCD DPMS and power */
#define DLCAPS_DPMS0            0x00000001  /* Can do DPMS state 0 */
#define DLCAPS_DPMS1            0x00000002  /* Can do DPMS state 1 */
#define DLCAPS_DPMS2            0x00000004  /* Can do DPMS state 2 */
#define DLCAPS_DPMS3            0x00000008  /* Can do DPMS state 3 */

#define DLCAPS_POWER0           0x00000001  /* Can do OnNow power state 0 */
#define DLCAPS_POWER1           0x00000002  /* Can do OnNow power state 1 */
#define DLCAPS_POWER2           0x00000004  /* Can do OnNow power state 2 */
#define DLCAPS_POWER3           0x00000008  /* Can do OnNow power state 3 */

/* TV format (connectivity) */
#define DLCAPS_TV_FMT_CVBS              0x00000001  /* CVBS on phono */
#define DLCAPS_TV_FMT_CVBS_ON_LUMA      0x00000002  /* CVBS on minidin Y-pin */
#define DLCAPS_TV_FMT_CVBS_ON_CHROMA0x00000004  /* CVBS on minidin C-pin */
#define DLCAPS_TV_FMT_SVIDEO            0x00000008  /* SVIDEO on mini-din */
#define DLCAPS_TV_FMT_RGB               0x00000010  /* RGB on 25-way DType */
#define DLCAPS_TV_FMT_SOG               0x00000020  /* Sync-on-grn on DType */
#define DLCAPS_TV_FMT_RGBS              DLCAPS_TV_FMT_RGB

/* TV system */
#define DLCAPS_TV_SYS_UNDERSCAN         0x00000001  /* Underscan */
#define DLCAPS_TV_SYS_OVERSCAN          0x00000002  /* Overscan */
#define DLCAPS_TV_SYS_SQPIX             0x00000004  /* Square pixel timing */
#define DLCAPS_TV_SYS_CCIR601           0x00000008  /* CCIR601 pixel timing */
#define DLCAPS_TV_SYS_PAL               0x00000010  /* PAL-BDGHI */
#define DLCAPS_TV_SYS_PAL_M             0x00000020  /* PAL-M */
#define DLCAPS_TV_SYS_PAL_N             0x00000040  /* PAL-N */
#define DLCAPS_TV_SYS_PAL_N_DASH        0x00000080  /* PAL-N' */
#define DLCAPS_TV_SYS_NTSC              0x00000100  /* NTSC-M */
#define DLCAPS_TV_SYS_NTSC_J            0x00000200  /* NTSC-J */

/* LCD geometry */
#define DLCAPS_LCD_GEO_ZOOM             0x00000001  /* Zoom functionality */
#define DLCAPS_LCD_GEO_LANDSCAPE        0x00000002  /* Portrait/landscape */

/*****************************************************************************
 DLMANDevCaps() structs
*****************************************************************************/
typedef struct tagCRTCAPS
{
   DWORD dwDPMS;            /* DPMS power down caps */
   DWORD dwPower;           /* 'OnNow' power management caps */
}
CRTCAPS, *PCRTCAPS;

typedef struct tagTVCAPS
```

```
{
  DWORD dwPower;          /* 'OnNow' power management caps */
  DWORD dwFormat;         /* RGB, SOG, SVIDEO, CVBS_X, ... */
  DWORD dwSystem;         /* PAL, NTSC, Overscan, CCIR601, ... */
}
TVCAPS, *PTVCAPS;

typedef struct tagLCDCAPS
{
  DWORD dwDPMS;           /* DPMS power down caps */
  DWORD dwPower;          /* 'OnNow' power management caps */
  DWORD dwGeometry;       /* Zoom, Portrait / Landscape */
}
LCDCAPS, *PLCDCAPS;

typedef struct tagUNITCAPS
{
   DWORD dwScreen;        /* Can do Gamma, BCSH, Sharp, Flicker, */
                          /* Pan, Position, Size, DPMS, ... */
   DWORD dwWindow;        /* Can Overlay, CKey, ...     */
  union
  {
    CRTCAPS sCRTCaps; /* CRT specific caps */
    TVCAPS  sTVCaps;  /* TV  specific caps */
    LCDCAPS sLCDCaps; /* LCD Specific caps */
  };
}
UNITCAPS, *PUNITCAPS;
```

3.18. Detect Output Device Presence

```
DWORD DLMANSense(PDMUNITCTL psDUC, PUNITSENSE psSense)

PURPOSE        To detect the monitor-attached status of a given display
               unit.

PARAMETERS     psDUC     - pointer to units DMUNITCTL structure.
               psSense   - pointer to UNITSENSE structure.

RETURNS        DL_OK, if successful else error code.
```

This API senses whether an output device is attached to the unit. For TV devices, the attached screen's expected input signal type (i.e. system and format) is also sensed and returned using the DLCAPS_XXX defines detailed above. Note that sensing may cause screen disturbance.

```
/*****************************************************************************
 Definitions for display unit attached in DLMANSense()
*****************************************************************************/
#define     DL_UNIT_SENSED    0x01

/*****************************************************************************
 DLMANSense() structs
*****************************************************************************/
```

```
typedef struct tagCRTSENSE
{
      DWORD dwSensed;          /* Monitor Sensed */
}
CRTSENSE, *PCRTSENSE;

typedef struct tagTVSENSE
{
      DWORD dwSensed;          /* TV Sensed */
      DWORD dwConnect;         /* RGBS, SVIDEO, CVBS, CVBS on Y, CVBS on C*/
}
TVSENSE, *PTVSENSE;

typedef struct tagLCDSENSE
{
      DWORD dwSensed;          /* LCD Sensed */
}
LCDSENSE, *PLCDSENSE;

typedef union tagUNITSENSE
{
      CRTSENSE sCRTSense; /* CRT specific sensed state */
      TVSENSE  sTVSense;  /* TV  specific sensed state */
      LCDSENSE sLCDSense; /* LCD Specific sensed state */
}
UNITSENSE, *PUNITSENSE;
```

3.19. Reset I2C

```
DWORD DLMANResetI2C(PDMUNITCTL psDUC)
```

PURPOSE — Performs a reset for the indicated I2C bus on the indicated master device.

PARAMETERS — psDUC - indicates which PMX I2C bus to reset.

RETURNS — DL_OK, if successful else error code.

This API should be called prior to a group of I2C accesses and in response to error conditions reported thereby. This API effectively writes a Stop condition to the indicated bus. The PMX 'A' or 'B' I2C bus is inferred by the API from the psDUC parameter. For CRT devices DLMAN uses the 'A' bus, for TV decoder devices, DLMAN uses the 'B' bus. DLMAN uses internal, lower-level, I2C functions to communicate with any external TV encoder or LCD devices, (for initialisation, set-up and sensing) and addresses the 'B' bus for these devices.

3.20. Write I2C

```
DWORD DLMANWriteI2C(PDMUNITCTL psDUC, BYTE bSlave, PBYTE pbBuf,
                                                    DWORD dwCount)
```

PURPOSE — Writes the contents of the indicated buffer to the I2C slave device connected to the indicated port on the master device.

```
PARAMETERS      psDUC      - indicates which PMX I2C bus to write.
                bSlave     - The I2C slave address of the target device.
                pbBuf      - points to the buffer of data to be written.
                dwCount    - is a count of the bytes to write.

RETURNS         DL_OK, if successful else error code.
```

3.21. Read I2C

```
DWORD DLMANReadI2C(PDMUNITCTL psDUC, BYTE bSlave, PBYTE pbBuf,
                                                        DWORD dwCount)

PURPOSE         Reads bytes from the I2C slave device connected to the
                indicated port on the master device to the indicated buffer.

PARAMETERS      psDUC      - indicates which PMX I2C bus to read.
                bSlave     - The I2C slave address of the target device.
                pbBuf      - points to the buffer to receive read data.
                dwCount    - is a count of the bytes to read.

RETURNS         DL_OK, if successful else error code.
```

3.22. Start I2C

```
DWORD DLMANStartI2C(PDMUNITCTL psDUC)

PURPOSE         Performs a start for the indicated I2C bus on the indicated
                master device.

PARAMETERS      psDUC      - indicates which PMX I2C bus to access.

RETURNS         DL_OK, if successful else error code.
```

This API should be called prior to a group of I2C accesses. This API effectively writes a Start condition to the indicated bus.

3.23. Stop I2C

```
DWORD DLMANStopI2C(PDMUNITCTL psDUC)

PURPOSE         Performs a stop for the indicated I2C bus on the indicated
                master device.

PARAMETERS      psDUC      - indicates which PMX I2C bus to access.

RETURNS         DL_OK, if successful else error code.
```

This API should be called after a group of I2C accesses. This API effectively writes a Stop condition to the indicated bus.

3.24. Get Display List ID

```
DWORD DLMANGetDListID(PDMUNITCTL psDUC, PDWORD pdwDLID)
```

```
PURPOSE         Retrieves the display list ID (DLID) for the display list
                associated with the primary display surface.

PARAMETERS      psDUC      - pointer to units DMUNITCTL structure
                pdwDLID    - address to receive the display list ID.

RETURNS         DL_OK, if successful else error code.
```

This API is intended for use by DirectX for the retrieval of the DLID for the display list associated with the primary surface which has already been created by the display driver via **DLMANDisplayMode()**.

3.25. Create Display List

```
DWORD DLMANCreateDList(PDMUNITCTL psDUC, PCREATEDL psCreate)

PURPOSE         Instantiates a display list with the resolution, cursor
                parameters and h/w timing parameters of the primary
                surface's display list. The list does not inherit the
                primary lists' video windows. The display list is not
                written to Frame Buffer, i.e. is not executed, until the
                DLMANAttachDList() API is called.

PARAMETERS      psDUC      - identifies the unit.
                psCreate   - specifies the creation parameters.

RETURNS         DL_OK, if successful else error code.
```

This API is intended for use by DirectX in the establishment of a display list for a back buffer. The **DLMANDisplayMode()** API establishes the primary display lists, the DLIDs for which are returned by **DLMANGetDListID()**. The CREATEDL struct is defined as:-

```
typedef struct tagCREATEDL
{
      DWORD        dwDLID;      /* DList ID returned by call */
      DWORD        dwPhysAddr;  /* PMX physical address of surface memory */
      BYTE         bSurf;       /* surface number (0 - 2) */
}
CREATEDL, *PCREATEDL;
```

3.26. Attach Display List

```
DWORD DLMANAttachDList (PDMUNITCTL psDUC, DWORD dwDLID)

PURPOSE         Writes the indicated display list to Frame Buffer. Further
                modifications of this 'attached' dlist (via modification of
                already attached overlays, attachment of new overlays) are
                reflected in the executing dlist in frame buffer.

PARAMETERS      psDUC identifies the unit for which the command is relevent.
                dwDLID is the display list ID to attach.

RETURNS         DL_OK, if successful else error code.
```

This API is intended for use by DirectX. Once attached, the display list may be 'flipped to', causing display of the back buffer and associated overlay surfaces it describes.

3.27. Detach Display List

```
DWORD DLMANDetachDList(PDMUNITCTL psDUC, DWORD dwDLID)

PURPOSE         Removes the dlist from the frame buffer.

PARAMETERS      psDUC identifies the unit for which the command is relevent.
                dwDLID is the display list ID to detach.

RETURNS         DL_OK, if successful else error code.
```

This API is intended for use by DirectX. It is inadvisable to call this function with the DLID of the currently displayable list.

3.28. Destroy Display List

```
DWORD DLMANDestroyDList(PDMUNITCTL psDUC, DWORD dwDLID)

PURPOSE         Destroys the display list.

PARAMETERS      psDUC identifies the unit for which the command is relevent.
                dwDLID is the display list ID to destroy.

RETURNS         DL_OK, if successful else error code.
```

This API is intended for use by DirectX. Note that all overlays solely associated with this DLID will be destroyed at this time. It is inadvisable to call this function with the DLID of the primary surface's display list.

3.29. Create Buffer

```
DWORD DLMANCreateBuffer(PDMUNITCTL psDUC, PCREATEBUF psCreate)

PURPOSE         Creates internal DLMAN structures that define the source
                data for a video window (overlay)

PARAMETERS      psDUC     - pointer to units DMUNITCTL structure.
                psCreate  - pointer to buffer creation structure.

RETURNS         DL_OK, if successful else error code.
```

The API is intended for use by DirectX. An overlay (video window) can be multiply buffered. This API specifies the buffer of source data comprising the video overlay. Buffers are attached to overlays as overlays are attached to display lists. An on-screen video window has a one-to-one correspondence with a DLMAN overlay and one display list can handle four, auto-flippable overlays, each of which can be quadruply buffered. The limitation of four quadruply-buffered video windows applies only to auto-flipping. DLMAN itself applies no inherent limit (except that of display and host RAM and the current size of the bSurf, bStream and bIndex members of the various creation structures) to the number of buffers or video windows. The CREATEBUF struct is shown below.

```
typedef struct tagCREATEBUF
{
```

```
    DWORD       dwBID;      /* Buffer ID returned by call */
    DWORD       dwFBPhys;   /* PMX phys addr of buffer data */
    DWORD       dwXExt;     /* pixel width of buffer data */
    DWORD       dwYExt;     /* number lines of buffer data */
    DWORD       dwStride;   /* pixel stride of buffer data */
    BYTE        bIndex;     /* buffer number (0 - 3) */
}
CREATEBUF, *PCREATEBUF;
```

3.30. Attach Buffer

```
DWORD DLMANAttachBuffer(PDMUNITCTL psDUC, DWORD dwOID, DWORD dwBID)

PURPOSE         Associates a previously created buffer with a video window
                (overlay).

PARAMETERS      psDUC     - pointer to units DMUNITCTL structure.
                dwOID     - Overlay ID to attach to.
                dwBID     - Buffer ID to attach to overlay.

RETURNS         DL_OK, if successful else error code.
```

The API is intended for use by DirectX. An overlay (video window) can be multiply buffered. This API associates the buffer of source data comprising the video overlay with the overlay and if all other attachments (overlay to dlist and dlist to device) have been made either creates/modifies multiple display lists (to reflect any multiple buffering and to provide for auto-flipping) or modifies/creates a single dlist in the simple case of a single overlay window with one buffer of source data.

3.31. Detach Buffer

```
DWORD DLMANDetachBuffer(PDMUNITCTL psDUC, DWORD dwOID, DWORD dwBID)

PURPOSE         Creates internal DLMAN structures that define the source
                data for a video window (overlay)

PARAMETERS      psDUC     - pointer to units DMUNITCTL structure.
                dwOID     - Overlay ID to detach from.
                dwBID     - Buffer ID to detach.

RETURNS         DL_OK, if successful else error code.
```

The API is intended for use by DirectX. This API removes a previous buffer/overlay association, causing modification and/or removal of single or multiple display lists.

3.32. Destroy Buffer

```
DWORD DLMANDestroyBuffer(PDMUNITCTL psDUC, DWORD dwBID)

PURPOSE         Creates internal DLMAN structures that define the source
                data for a video window (overlay)
```

```
PARAMETERS      psDUC      - pointer to units DMUNITCTL structure.
                dwBID      - Buffer ID to destroy.

RETURNS         DL_OK, if successful else error code.
```

The API is intended for use by DirectX. This API removes a previous buffer/overlay association, causing modification and/or removal of single or multiple display lists.

3.33. Create Overlay

```
DWORD DLMANCreateOverlay(PDMUNITCTL psDUC, PCREATEOVL psCreate)

PURPOSE         Instantiates overlay (video window) data.

PARAMETERS      psDUC      - pointer to units DMUNITCTL structure
                psCreate   - pointer to create overlay structure

RETURNS         DL_OK, if successful else error code.
```

This API is intended for use by DirectX. The CREATEOVL struct is defined as:-

```
typedef struct tagCREATEOVL
{
      DWORD dwOID;       /* Overlay ID returned by call */
      DWORD dwFlags;     /* bring-to-top, visible, bob/weave, field/frame */
      LONG  lDstX1;      /* On-Screen Left pixel pos of new window */
      LONG  lDstY1;      /* On-Screen Top pixel pos of new window */
      LONG  lDstX2;      /* On-Screen Right pixel pos of new window */
      LONG  lDstY2;      /* On-Screen Bottom pixel pos of new window */
      LONG  lSrcX1;      /* Source Data Left pixel pos of new window */
      LONG  lSrcY1;      /* Source Data Top pixel pos of new window */
      LONG  lSrcX2;      /* Source Data Right pixel pos of new window */
      LONG  lSrcY2;      /* Source Data Bottom pixel pos of new window */
      DWORD dwPixFmt;    /* pixel format see DOC for encoding */
      BYTE  bStream;     /* stream number for this overlay (0 - 3) */
}
CREATEOVL, *PCREATEOVL;

/* CREATEOVL and ATTROVL dwFlags options */
#define OVL_FLAGS_VIS   0x0001            /* visible or invisible */
#define OVL_FLAGS_TOP   0x0002            /* topmost or not topmost */
#define OVL_FLAGS_FLD   0x0004            /* field or frame */
#define OVL_FLAGS_BOB   0x0008            /* bob or weave */


/* CREATEOVL and ATTROVL dwPixFmt options */
#define OVL_RGB8              0x00        /* 8bit indexed colour */
#define OVL_RGB15                   0x01        /* 5.5.5 RGB */
#define OVL_RGB16                   0x02        /* 5.6.5 RGB */
#define OVL_RGB24                   0x03        /* 8.8.8 RGB */
#define OVL_RGB32                   0x04        /* 8.8.8.8 aRGB */
#define OVL_YUYV              0x05        /* 4.2.2 YUV (YUYV) */
#define OVL_YUV               0x06        /* 4.4.4 YUV */
#define OVL_UYVY              0x07        /* 4.2.2 YUV (UYVY) */
```

3.34. Attach Overlay

```
DWORD DLMANAttachOverlay(PDMUNITCTL psDUC, DWORD dwDLID, DWORD dwOID)

PURPOSE        Creates a new display list to reflect the additional overlay
               surface (video window).

PARAMETERS     psDUC      - pointer to units DMUNITCTL structure
               dwDLID     - display list identifier to attach to.
               dwOID      - overlay identifier to be attached.

RETURNS        DL_OK, if successful else error code.
```

This API is intended for use by DirectX. DLMAN modifies the indicated display list to reflect the new overlay surface. If the indicated overlay surface is invisible (dwFlags member of overlay struct != OVL_ATTR_VIS) no change in display list occurs else both host and frame buffer copies (if dlist attached to display) are updated to reflect the attachment.

3.35. Detach Overlay

```
DWORD DLMANDetachOverlay(PDMUNITCTL psDUC, DWORD dwDLID, DWORD dwOID)

PURPOSE        Modifies the dlist in respect of the video window data.

PARAMETERS     psDUC      - pointer to units DMUNITCTL structure.
               dwDLID     - is the display list to detach from.
               dwOID      - is the overlay identifier to be detached.

RETURNS        DL_OK, if successful else error code.
```

This API is intended for use by DirectX. This command has the effect of modifying the associated display list for this buffer. If the indicated dlist is already attached to a display the dlist change is reflected immediately in the frame buffer copy of the dlist.

3.36. Destroy Overlay

```
DWORD DLMANDestroyOverlay(PDMUNITCTL psDUC, DWORD dwOID)

PURPOSE        removes the instantiation of the video window data.

PARAMETERS     psDUC      - pointer to units DMUNITCTL structure.
               dwOID      - is the overlay ID of the overlay to destroy.

RETURNS        DL_OK, if successful else error code.
```

This API is intended for use by DirectX. This API does not destroy the others of a multiply buffered set of overlays, this must be done explicitly

3.37. Set Overlay Attributes

```
DWORD DLMANSetOverlayAttributes(PDMUNITCTL psDUC, PATTROVL psAttr)

PURPOSE        Modifies the indicated display list to reflect requested
               change in overlay data.
```

```
PARAMETERS      psDUC      - pointer to unit's DMUNITCTL structure.
                psAttr     - is a pointer to the structure determining the
                             overlay attribute change.

RETURNS         DL_OK, if successful else error code.
```

This API is intended for use by DirectX. Dlists already attached to displays and to which the indicated overlay is attached are updated in both host copy and frame buffer with immediate visible effect. The ATTROVL struct is defined as:-

```
typedef struct tagATTROVL
{
      DWORD dwOID;       /* Overlay ID returned by call to CreateOverlay */
      DWORD dwAttrSpec; /* bitmask of attrib(s) to set */
      DWORD dwFlags;     /* bring-to-top, visible, bob/weave, field/frame */
      LONG  lDstX1;      /* On-Screen Left pixel pos of new window */
      LONG  lDstY1;      /* On-Screen Top pixel pos of new window */
      LONG  lDstX2;      /* On-Screen Right pixel pos of new window */
      LONG  lDstY2;      /* On-Screen Bottom pixel pos of new window */
      LONG  lSrcX1;      /* Source Data Left pixel pos of new window */
      LONG  lSrcY1;      /* Source Data Top pixel pos of new window */
      LONG  lSrcX2;      /* Source Data Right pixel pos of new window */
      LONG  lSrcY2;      /* Source Data Bottom pixel pos of new window */
      DWORD dwPixFmt;    /* pixel format see DOC for encoding */
}
ATTROVL, *PATTROVL;
```

The dwPixFmt and dwFlags members of ATTROVL take the same values as the corresponding members of the CREATEOVL struct defined above.

```
/* ATTROVL dwAttrSpec options */
#define OVL_ATTR_FLAGS  0x0001
#define OVL_ATTR_DSTX1  0x0002
#define OVL_ATTR_DSTY1  0x0004
#define OVL_ATTR_DSTX2  0x0008
#define OVL_ATTR_DSTY2  0x0010
#define OVL_ATTR_SRCX1  0x0020
#define OVL_ATTR_SRCY1  0x0040
#define OVL_ATTR_SRCX2  0x0080
#define OVL_ATTR_SRCY2  0x0100
#define OVL_ATTR_PIXFMT 0x0200
```

3.38. Get/Set Gamma

```
DWORD DLMANGamma(PDMUNITCTL psDUC, DWORD dwFlags, PLONG plGamma)

PURPOSE         Set or Get the gamma value for this display unit.

PARAMETERS      psDUC      - pointer to units DMUNITCTL structure.
                dwFlags    - Set/get default/new gamma value.
                plGamma    - pointer to long containing gamma value.

RETURNS      DL_OK, if successful else error code.
```

The gamma value supplied to this API is a single precision floating point number encoded in a long. The function generates an appropriate palette and programs the hardware CLUT(s) to

establish the gamma correction. An appropriate error is returned if this API is used with display units which do not support the functionality. Definitions are exported for the allowable range of input gamma values:-

```
/*****************************************************************************
 DLMANGamma() API range defines
*****************************************************************************/
#define DL_MAX_GAMMA 4.00
#define DL_MIN_GAMMA 0.01
```

3.39. Get/Set Palette

```
DWORD DLMANPalette(PDMUNITCTL psDUC, DWORD dwFlags, DWORD dwIndex,
                                          DWORD dwCount, PBYTE pbPal)

PURPOSE        Set or Get the GDI colour palette entry(ies) for this
               display unit.

PARAMETERS     psDUC     - pointer to units DMUNITCTL structure.
               dwFlags   - Set/get default/new palette entries.
               dwIndex   - start index for palette entry writes/reads
               dwCount   - count of entries to read/write.
               pbPal     - buffer to send or receive palette entry data.

RETURNS      DL_OK, if successful else error code.
```

Called predominantly by the 2D display driver miniport/miniVDD, this API takes an input palette of the form of a GDI palette and writes it to the hardware CLUT of the indicated device. An appropriate error is returned if this API is used with display units which do not support the functionality.

3.40. Get/Set Colour Key

```
DWORD DLMANCKey(PDMUNITCTL psDUC, DWORD dwFlags, PCKREGS psRegs)

PURPOSE        Set or Get the colour/chromakey value for this display unit.

PARAMETERS     psDUC     - pointer to units DMUNITCTL structure.
               dwFlags   - Set/get default/new colour key regs.
               psRegs    - pointer to structure defining colour key regs.

RETURNS      DL_OK, if successful else error code.
```

This API is expected to be used by DirectX. The dwFlags member of CKREGS indicates whether the data to set/get is a chromakey or colourkey value. The overlay hardware is programmed with the incoming data and set to the indicated mode of operation. An appropriate error is returned if this API is used with display units which do not support the functionality.

```
/*****************************************************************************
 Definitions for dwFlags member of CKREGS struct in DLMANCKey() API
*****************************************************************************/
#define CK_COLOURKEY    0
#define CK_CHROMAKEY    1
```

```
/*****************************************************************************
 CKREGS struct definition for DLMANCKey() API
*****************************************************************************/
typedef struct tagCKREGS
{
      DWORD dwFlags;
      union tagUCK
      {
          struct
          {
              DWORD dwColourKey;
          };
          struct
          {
              BYTE bB;
              BYTE bG;
              BYTE bR;
              BYTE bX;
          };
          struct
          {
              DWORD dwChromaKey;
          };
          struct
          {
              WORD wU2U1;
              WORD wV2V1;
          };
          struct
          {
              BYTE bU1;
              BYTE bU2;
              BYTE bV1;
              BYTE bV2;
          };
      }
      UCK;
}
CKREGS, *PCKREGS;
```

3.41. Get/Set Picture Quality

```
DWORD DLMANPicture(PDMUNITCTL psDUC, DWORD dwFlags, PPQUAL psPQual)

PURPOSE        Set or Get the picture quality regs for this display unit.

PARAMETERS     psDUC     - pointer to units DMUNITCTL structure.
               dwFlags   - Set/get default/new picture quality regs.
               psPQual   - pointer to structure defining quality regs.

RETURNS      DL_OK, if successful else error code.
```

This API effects control over picture quality, in terms of a soft or sharp image quality and allows selection of flicker filtering. An appropriate error is returned if this API is used with display units which do not support the functionality.

```
/*****************************************************************************
 DLMANPicture() structs
*****************************************************************************/
typedef struct tagPQUAL
{
      DWORD dwFilter;    /* Flicker-filter on/off */
      DWORD dwQuality;   /* 0=soft, 1=sharp */
      BYTE bFlicker[12];/* Packed FIR coeffs if flicker-filter is on */
}
PQUAL, *PPQUAL;

/*****************************************************************************
 Definitions for dwFilter member of PQUAL in DLMANPicture() API
*****************************************************************************/
#define PIC_FLICKER_OFF       0
#define PIC_FLICKER_ON        1

/*****************************************************************************
 Definitions for dwQuality member of PQUAL in DLMANPicture() API
*****************************************************************************/
#define PIC_SOFT              0
#define PIC_SHARP             1
```

3.42. Virtual Machine State-Change Callback

```
STATE_CALLBACK DLCBState(PDEVICE_CTLBLK psDCB, DWORD dwState, DWORD dwLoss)

PURPOSE      Receives notification from Kernel Manager of HiRes to VGA state
             changes. DLCBState is a STATE_CALLBACK function registered with
             Kernel Manager at InitDevice() time.

PARAMETERS   psDCB - KM device ID
             dwState - flags representing state change
             dwLoss - flags representing context loss severity

RETURNS      DL_OK, if successful else error code.
```

This API is not present in the DLMAN service table but is registered, at DLMAN initialisation, with the Kernel Manager and is included here for completeness. The **DLCBState()** function is called back by the Kernel Manager to inform DLMAN of fullscreen DOS-box to Windows SVGA mode transitions. The dwLoss flags word uses the following defines:

```
/*****************************************************************************
 How much context we lose on a mode change
*****************************************************************************/
#define PMX_NO_CONTEXT_LOSS                0
#define PMX_SEVERE_LOSS_OF_CONTEXT  1
```

The dwState flags word uses the following defines:

```
/*****************************************************************************
 Flag for internal use
```

```
*****************************************************************************/
#define PMX_PRE_STATE_CHANGE_MASK 0x80

/*****************************************************************************
 VGA Emulation-Native display mode state change defs. Used in DLMANState()
and KMState() APIs.
*****************************************************************************/
#define PMX_PRE_HIREZ_TO_VGA         0x1 | PMX_PRE_STATE_CHANGE_MASK
#define PMX_POST_HIREZ_TO_VGA        0x1
#define PMX_PRE_VGA_TO_HIREZ         0x2 | PMX_PRE_STATE_CHANGE_MASK
#define PMX_POST_VGA_TO_HIREZ        0x2

#define PMX_PRE_XY_REZ_CHANGE        0x3 | PMX_PRE_STATE_CHANGE_MASK
#define PMX_POST_XY_REZ_CHANGE       0x3
#define PMX_PRE_Z_REZ_CHANGE         0x4 | PMX_PRE_STATE_CHANGE_MASK
#define PMX_POST_Z_REZ_CHANGE        0x4
#define PMX_PRE_WINDOWS_MODE_CHANGE 0x9
```

3.43. Power State-Change Callback

```
STATE_CALLBACK DLCBPower(PDEVICE_CTLBLK psDCB, DWORD dwState, DWORD dwLoss)

PURPOSE      Receives notification from Kernel Manager of ACPI 'OnNow' Power
             management state changes. DLCBPower is a STATE_CALLBACK
             function registered with Kernel Manager at InitDevice() time.

PARAMETERS   psDCB - KM device ID
             dwState - flags representing state change
             dwLoss - flags representing context loss severity

RETURNS      DL_OK, if successful else error code.
```

This API is not present in the DLMAN service table but is registered with the Kernel Manager and is included here for completeness. The **DLCBPower()** function is called by the Kernel Manager to inform DLMAN of ACPI 'OnNow' power management state transitions. The dwLoss flags are as defined for **DLCBState()**. The dwState flags word uses the following defines:

```
/*****************************************************************************
 KM 'OnNow' power state API state defines
*****************************************************************************/
#define KM_POW_STATE0          0x5
#define KM_POW_STATE1          0x6
#define KM_POW_STATE2          0x7
#define KM_POW_STATE3          0x8

/*****************************************************************************
 Power state definitions used in DLMANPower() and KMPower() APIs.
*****************************************************************************/
#define PMX_PRE_POW_STATE0     KM_POW_STATE0 | PMX_PRE_STATE_CHANGE_MASK
#define PMX_POST_POW_STATE0    KM_POW_STATE0
#define PMX_PRE_POW_STATE1     KM_POW_STATE1 | PMX_PRE_STATE_CHANGE_MASK
#define PMX_POST_POW_STATE1    KM_POW_STATE2
```

```
#define PMX_PRE_POW_STATE2     KM_POW_STATE2 | PMX_PRE_STATE_CHANGE_MASK
#define PMX_POST_POW_STATE2    KM_POW_STATE2
#define PMX_PRE_POW_STATE3     KM_POW_STATE3 | PMX_PRE_STATE_CHANGE_MASK
#define PMX_POST_POW_STATE3    KM_POW_STATE3

/*****************************************************************************
 How much context we lose on a state change
*****************************************************************************/
#define PMX_NO_CONTEXT_LOSS          0
#define PMX_SEVERE_LOSS_OF_CONTEXT   1
```

4. PRIMARY CORE INTERFACE.

DLMAN interacts with the Primary Core through a common data stucture which is allocated when the code fragments are loaded via the Kernel Manager Load code fragment interface. The structure is defines as:

4.1. RTComms Structure.

```
typedef struct tagRTDISPLAYIF
{
      DWORD dwSense;     /* if non-zero is an auto-sense dlist address */
      DWORD dwStop;      /* set by host to stop thread, reset by thread */
      DWORD dwFlip;      /* b10 set/reset by DLMAN for DLMAN-type flip */
      DWORD dwResetFlip;/* Mask of which bits in T3FLIP_RQ are valid */
      DWORD dwCoreReg1Lo0;
      DWORD dwCoreReg1Hi0;
      DWORD dwCoreReg1Lo1;
      DWORD dwCoreReg1Hi1;
      DWORD dwStatus;    /* b0 set = VBLANK in progess else reset */
                         /* b1 set = thread running */
      DWORD dwVCount;    /* incrementing VSYNC count */
      DWORD dwFirstTime;/* Flag indicating mode settime */
      DWORD dwPadding;  /* Padding to keep the table quadword aligned*/
      DWORD adwDList[2][4][4][4][4][4];
                         /* Table of display lists*/
}
RTDISPLAYIF, *PRTDISPLAYIF;
```

4.11. dwSense.

The dwSense member can be used to display a specific display list whose physical address is passed via this member. This allows the setting of a known 50% grey display list that would

allow device sensing via the comparators, this display list overrides any other display lists and becomes active when the thread finishes the display list currently being processed.

4.12. dwFlip

The dwFlip member is used to flip between two halfs of the display list table, this is to allow us to make changes to a display list and have them take affect after the current display list is finished being processed, i.e. at vSync time.

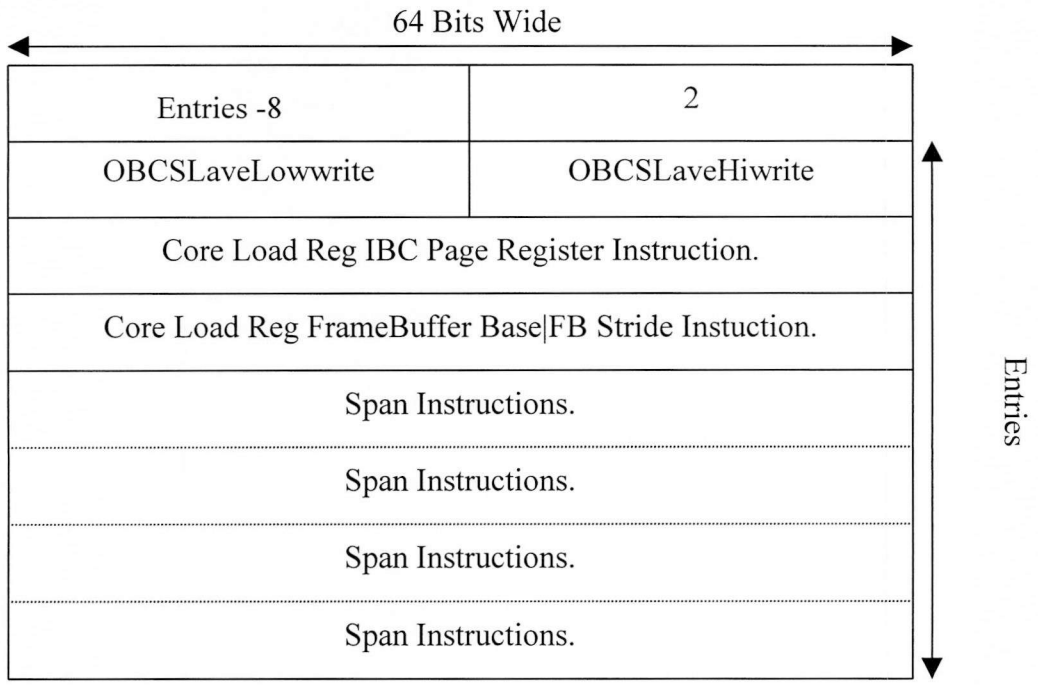
4.13. dwStop.

The dwStop is used to control display list processing, the Host can Set the stop flag to non zero and the Dac thread will stop display list processing and then clear the stop member. Note that after display list processing has stopped there may still be data in the Dacfeed fifo, which must be allowed to empty before any Dac registers are modified.

4.14. adwDList.

The adwDList table is a table of pointers to Display Lists as described in the PowerVR 250 Programming guide. The Display lists should take the format described below.

4.141. Display List Format.



The OBCSlave Low and high write entries are used by the DAC Thread and can be constructed as

```
((OBC_EVENT_WR | DAC_STREAM_0_ADDR_LO) << 16) | (dwSlaveBase & 0xFFFF)
```

and

```
((OBC_EVENT_WR | DAC_STREAM_0_ADDR_HI) << 16) | (dwSlaveBase >> 16)
```

4.142 adwTable Index Format.

The entry in the adwDList table is obtained from the properties of the display List , the index is built up by the active table index (1bit), surface number , and visible overlay numbers A,B,C and D, (2bits).

Active table	BufferA		BufferB		BufferC		BufferD		Surface	

The active table bit allows us to double buffer our table and thus allows seamless flipping of display lists by building the display list and putting the address into the inactive half of the table and setting the dwFlip member of the RTComms structure, this is read by DAC thread and takes affect when it next reads the Dlist. Only the Active table should be set in the RTComms structure as the DirectDraw Flip controls other entries.

4.15. dwResetFlip.

The dwResetFlip field is used to specify what are valid display lists, this is done to prevent accidentally trying to flip to non-valid display lists which will cause the DAC thread to fail. The mask bits represent the valid entries of the table and are encoded in the same way as the adwTable index as described above but without the addition of the active table bit .

4.16. dwStatus.

The dwStatus Member is used to determine the state of the DAC Thread, the thread can either be active or not, if it is active then bit 1 will be set. If we are in blanking or in reset then bit 0 will be set.

4.17. dwVCount.

The dwVCount Member is a count of the number of Vsyncs encountered, this will reset when it overflows. The host should clear this on mode change.

4.18. dwFirstTime.

The dwFirstTime member should be set to nonzero when starting the DAC after a mode change. If the DAC has been stopped to modify registers, then set this to zero.

4.19. dwCoreReg0Lo/Hi 1Lo/Hi

The Corereg members are used to load up the internal DAC Register 1 ,which controls cursor information, each time through a display list these members are read and sent to the DAC, both the HI members should always be set to 0x40000001, both the Lo members should be set to the desired value for the register.

5. DLMAN Operation Overview.

Here are detailed basic procedures for initialising DLMAN, setting up a mode and creating an overlay.

5.1. Initialising the Module.

DLMAN works in conjunction with the Kernel Manager and Display Driver at windows start-up time to initialise the Card . At boot time Windows calls the PMXKern PNPInitdevice function, which does chip initialisation and in turn will result in a call to DLMANInitdevice, which will create the required DUC pointers and allocate the RTComms structure with which it will communicate with the Primary Core. This will then load code fragments and start up the DAC thread. Once DUC's and DDC's have been allocated the DUC's must be assocaiated to a Unit syuch as a CRT or TV control then passes back to the Kernel Manager and Windows.

```
if (psDLM->DLMANInitDevice(1, 0, &psDDC) != DL_OK)
{
      DPF("Failed to init DL device\n");
      return (PMXDXSRV_ERROR_NODLM);
}
/* Associate DAC and TV devices to the unit */
if (psDLM->DLMANAssocOSDevToVLUnit(1, psDDC, DL_CRT_UNIT, &psDACDUC) !=
DL_OK)
{
      DPF("Failed to associate CRT unit to DLM\n");
      return (PMXDXSRV_ERROR_NODLM);
}
```

5.2. Setting the Mode.

The next major stage is when DLMAN is needed to set a display mode. This will occur through the Display Driver DisplayMode call, and will pass the DUC on which to set the Mode, a pointer to a DLMode structure and flags to say if this is a get or set mode, if it is a get mode then the structure is filled in with the current mode parameters. If a set mode is required then the parameters are used to set up the necessary registers and to set the mode.

```
/* Get default display mode refresh rate etc*/
if (psDLM->DLMANDisplayMode(psDACDUC, DL_GET_DEF, &sMode) != DL_OK)
{
      DPF("Failed to get default DAC mode\n");
      return (PMXDXSRV_ERROR_NODLM);
}
sMode.dwXExt=SCREEN_WIDTH;
sMode.dwYExt=SCREEN_HEIGHT;
sMode.dwStride=SCREEN_STRIDE;
sMode.dwBPP=SCREEN_BPP;
/* Set the mode with the above attributes */
if (psDLM->DLMANDisplayMode(psDACDUC, DL_SET, &sMode) != DL_OK)
{
      DPF("Failed to set DAC mode\n");
      return (PMXDXSRV_ERROR_NODLM);
}
```

5.3. Creating Overlays.

Overlays are created via the PMXDXSRV module, which acts as a layer of abstractions between the HAL and DLMAN. First the calling process must allocate the physical memory for the buffers it will use. This memory is then attached to a buffer via the DLMANCreateBuffer call, this will fill in a buffer structure with width, stride of buffer, and link this in with a list of allocated buffers. Next the Overlay is created via the DLMANCreatOvl call, this will set up an overlay with specified scaling pixel format position etc. These must then be attached to each other, this is achieved via the DLMANAttachBuffer call which will associate that buffer with the overlay. The CreateBuffer and AttachBuffer must be repeated for each buffer on that overlay. These overlays must then be attached to a surface, this is done via the DLMANAttachOverlay call which will link the Overlay to the given surface, if this overlay is created with the visible flag set then it will be displayed with the given attributes on screen. Overlays are then manipulated with the DLMANSetOverlayAttributes function, and can be used to mode, resize, and hide overlays. The overlays can be destroyed by first detaching the Overlays and Buffers in question using the DLMANDetachOverlay and DLMANDetachBuffer functions and then destroyed using the DLMANDestroyOverlay and DLMANDestroyBuffer functions.

```
if(AllocCardMem(psDriverData, psSurfLcl->ddsCaps.dwCaps,
                                        dwDisplaySurfaceSize,
                                        &psMCParams->psDisplayBufs[i],
                                        &dwPhysBase) != PMX_OK)
{
      hRes = DDERR_OUTOFMEMORY;
}
/* Create overlay buffer */
if(DLMANCreateBuffer(psDriverData, &sNewBuffer) != DL_OK)
{
      DPF("Could not create buffer for overlay");
      return FALSE;
}
if(DLMANCreateOverlay(psDriverData, &sNewOverlay) != DL_OK)
{
      DPF("Failed to create overlay");
      hRes = DDERR_GENERIC;
      goto CreateSurfaceError;
}


/*Attach buffer to overlay */
if(DLMANAttachBuffer(psDriverData, psOverlayInfo->dwOverlayID,
sNewBuffer.dwBID) != DL_OK)
// attach overlay to surface dwDListID
if(DLMANAttachOverlay(psDriverData, dwDListID, psNewOverlay->dwOverlayID)
!= DL_OK)
{
      DPF("Failed to attach overlay (0x%lx) to DList (0x%lx)",
      psNewOverlay->dwOverlayID, dwDListID);
      hRes = DDERR_GENERIC;
      goto CreateSurfaceError;;
}
```

5.4 Changing Mode.

Changing mode is achieved in much the same way as setting a mode except the DLCBState functions should be called with the PRE_WINDOWS_MODECHANGE state, this will clean up any outstanding display lists in preparation for the new mode.

6.PRIMARY CORE INTERFACE OPERATION OVERVIEW.

The Primary Core is not accessible to user mode applications and care should be taken to avoid causing it to get into an unrecoverable state and subsequent loss of display functionality.

When DLMAN initialises it loads the primary core with the DAC code fragment and launches the execute address in the same way as other modules.

When setting up a mode DLMAN must first stop the DAC thread processing display lists, by writing to the dwStop member of RTComms, to allow DAC registers to be changed to match the new display mode requirement. The members of the RTComms structure should be set to zero with the exception of the adwDLTab which will already have been cleared by the PMX_PRE_WINDOWS_MODECHANGE state call-back. It is then safe to create display lists and modify registers before starting up the thread by hitting the kicker.

```
DLISTCreateSurf(psDUC, psURegs, &dwSID, psMode->dwPhysAddr, 0xFF);
psDUC->psRTComms->dwSense = 0;
psDUC->psRTComms->dwStop = 0;
psDUC->psRTComms->dwStatus = 0;
psDUC->psRTComms->dwFirstTime=1;
psDUC->psRTComms->dwVCount=0;

*(psDUC->sDLCmdIF.pdwExeAddr) = psDUC->sDLCmdIF.dwProc;
/* and go... */
*(psDUC->sDLCmdIF.pdwKicker) = 1;
```

Display lists can be switched by building the display list, and putting it in framebuffer memory. A pointer to the display list is then put in the inactive half of the table, which is then flipped to by writing to the dwFlip member, after waiting until the flip has taken place ,i.e. dwVCount has changed twice (not once as it will take time for the flip to be seen and dwVCount may change before the flip has been seen) , it is then safe to update the inactive half of the table. Unless a change in the slave stream registers or PLL is required this is the

best way to modify a display, if the slave stream does need adjusting as in the case of adding an overlay then it is necessary to stop the DAC thread before modifying the registers.

```
GenChipDL(psDUC, psCtl, psDUC->bActiveDLTab);
psDUC->bActiveDLTab ^= 1;              /* flip to new list */
FENCED_WRITE ((DWORD)&psDUC->psRTComms->dwFlip, (DWORD)psDUC->bActiveDLTab
<< 10);
WaitVCountChange(psDuc->psRTComms,TimeOut);
WaitVCountChange(psDuc->psRTComms,TimeOut);
```

To stop the DAC a non-zero value should be written to the dwStop member of RTComms, it is then advisable to allow a delay of about 50 milliseconds for the DAC fifos to empty and the dacfeed to shutdown before modifying any registers.

```
psDUC->psRTComms->dwStop=1;
Wait(50); // wait 50 milliseconds to allow the DAC fifos to empty
ModifyRegs(psURegs);
DLISTCreateSurf(psDUC, psURegs, &dwSID, psMode->dwPhysAddr, 0xFF);
psDUC->psRTComms->dwSense = 0;
psDUC->psRTComms->dwStop = 0;
psDUC->psRTComms->dwStatus = 0;
psDUC->psRTComms->dwFirstTime=0;
psDUC->psRTComms->dwVCount=0;

*(psDUC->sDLCmdIF.pdwExeAddr) = psDUC->sDLCmdIF.dwProc;
/* and go... */
*(psDUC->sDLCmdIF.pdwKicker) = 1;
```